Creating Requirements-Based Estimates Before Requirements Are Complete

Carol A. Dekkers
Quality Plus Technologies, Inc.

Despite advances in tools and techniques, it is interesting to note that on-time and on-budget projects account for a mere one-third of projects today. While overly optimistic estimates are part of the problem, missing and incomplete requirements, and poor estimating methods share the blame. Accurate estimating is further challenged when customers demand estimates before requirements development begins.

Project estimating is formidable from the start – especially during/before the requirements discovery process. Poor requirements lead to poor estimates and poor schedules. Subsequently, changes are difficult to assess when the requirements are poor. Sometimes a service request ends up deferred to the next release due to confusion about requirements – and the next release fares no better. This leads to classic project failure – over budget and behind schedule – similar to the two-thirds of projects cited in the 2003 Standish Group's Chaos report. [1]

Estimators can start the process by determining how big and how complex is the user problem, how hard it will be to build, and how much confidence is needed in the estimate. Most often, however, we do not estimate this way. We start with a seemingly arbitrary end date and then count backwards to get the schedule, cost, and resources that can fit into the timeframe. Called *date-driven estimating* by author Steve McConnell, it is the most commonly used method. [2]

To complicate matters, date-driven estimates are usually task-based and rely on the *experience* and mental models of team members or cost estimators. Problems emerge on new projects with new technology or new subject matter where there is no prior experience from which to draw. An estimator is forced to seek other data on which to base an estimate.

Sophisticated parametric-based estimating models such as COCOMO II, SLIM, and SEER/SEM serve to provide the missing data with databases or proven industry equations. In most cases, however, some form of project size is a required input variable, along with other variables covering functional, quality, design, and technical drivers. Because any estimate is only as accurate as its least accurate input variable, we should not be surprised when projects exceed estimates for cost, schedule and duration. The Standish Group report [1] proclaimed a mere 33 percent of projects a success; however, this is double the results a mere decade ago.

As one of the first authors to recognize that software engineering differs from traditional engineering, David Card stated, "Engineering projects usually can wait until after design to provide an estimate, while software engineering requires an estimate before design" [3].

In the author's experience, software projects can be even worse – some projects need estimates before requirements! If we are to increase information technology (IT) credibility, we need to figure out ways to create auditable and reliable project estimates from initial project realization all the way through to project completion. One of the best ways to do this is to augment our current estimating method(s) with at least one requirements-based estimate. This additional approach serves to validate or invalidate the other estimate(s) and ensures that at least one method considered the size of the problem as an important project estimating variable.

Requirements Demystified

Given that project requirements are the source of 60 percent to 99 percent of defects delivered into production [4], and that project size based on requirements is a key input driver for project estimates [5], it makes sense to examine what can be done to clarify and further exploit the discovery of complete requirements early in the project.

The requirements discovery and articulation process should strive to maximize the known requirements while managing to minimize the unknowns. To clarify project requirements, divide them into three types: functional, non-functional, and technical requirements, as outlined in the following sections.

Functional Requirements

This type of requirements represents the unit work processes performed or supported by the software, (e.g., software for an altimeter records the ambient temperature). These requirements are part of the users'[1]/customers' responsibility to define, even though they may abdicate the initial specifications to the development team. Functional requirements can be thought of similar to a software floor plan – they are independent of any design constraints or technical implementation. Functional requirements can be documented with use cases and sized using functional size measurement (function points).

Once the functional requirements are sized, and other project requirements are known (see non-functional and technical requirements), cost estimates can be prepared using a Project Cost Ratio for comparable completed projects (see Table 1).

Non-Functional Requirements

This type of requirements represents how the software must perform once it is built. Also referred to as quality requirements, these requirements address the *ilities*: (suitability, accuracy, interoperability, compli-

Table 1: *Project Requirements Size-Based Estimating Equations*

Metric	Units	Equation
Project Cost Ratio (completed projects)	$/Function Point (FP)	Project Cost Rate = (Total Hours x Hourly Cost) + Other Costs / Project Functional Size
Annual Support Cost Ratio	Actual Support Costs per 1,000 FP (or Full Time Resources/Application)	Support Cost Ratio = (Yearly Support Hours x Hourly Cost) + Other Costs / Application Functional Size
Repair Cost Ratio	$/FP (or per fix)	Repair Cost Ratio = (Repair Hours x Hourly Cost) / Functional Size of Repair

ance, security, reliability, efficiency, maintainability, portability, and quality in use) as described by ISO [International Organization for Standardization] standards in [7] and performance criteria.

More often, non-functional requirements are discussed only at a high level and are often found scattered throughout various requirements documents. Using a construction analogy, the non-functional requirements are like the *contracted specifications* for software and outline the necessity for data accuracy (e.g., trajectory systems), response time (e.g., service level agreements), security (e.g., encryption), performance (e.g., 24x7 operation with replicated databases to prevent data loss), etc.

Technical (Build) Requirements

These project requirements are defined by how the software will be *built* to satisfy the functional and non-functional requirements. Technical requirements include the physical implementation characteristics of the project and include, for example, programming language, Computer-Aided Software Engineering (CASE) or other tools, methods, work-breakdown structure, type of project, etc. In practice, it is the technical requirements that document the design, and with the functional and non-functional requirements give rise to project specifics like Gantt charts, development methodology, reuse, etc. Technical requirements are to software as plumbing is to building construction.

All three types of project requirements are necessary to do a realistic project estimate. Functional size measurement strictly pertains only to the size of the software's functional user requirements.

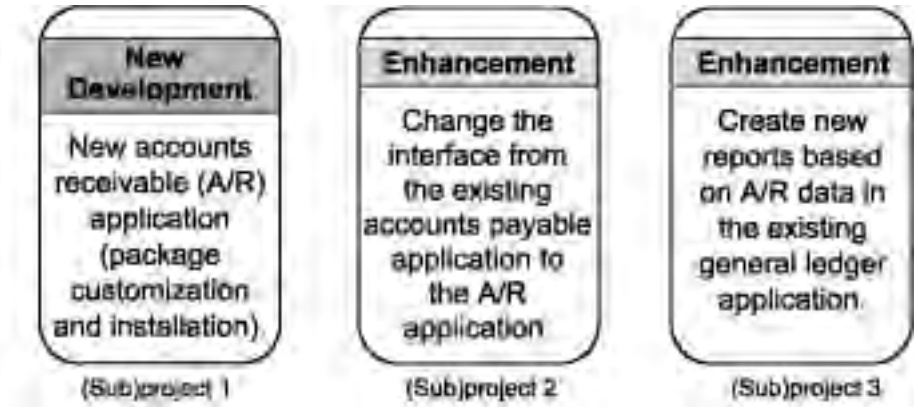
Modern software development approaches such as use cases and agile development attempt to categorize and keep these three types of requirements distinct and separate. Unfortunately in a manner similar to the contractor who only has a hammer and everything looks like a nail, some software developers cannot overcome the need to insert technical requirements into modern method deliverables such as use cases and agile user stories.

Estimating Challenges

The more information you know before making an estimate, the better the estimate should be. However, estimating faces challenges even with skilled estimators and high-quality teams. A few challenges include these: accuracy of input values (size, complexity, technical requirements, etc); availability of input variables; applicability of historical databases; completeness of the requirements (including functional, non-functional, and technical); tasks to be included; and risk factors. In spite of the challenges, cost estimators do produce estimates of duration, cost, and effort, which are turned into project schedules. Estimates made early in the development life cycle face large variations because of uncertainty. Estimates based on guessed input values are unreliable, yet many managers treat them as predictive project forecasts. We can alleviate this problem with a few guidelines: *Frame the guesstimate (an estimated guess) as preliminary.* When providing a guesstimate, frame it as a range of values (e.g., based on assumptions, the project could cost $250,000 to $600,000). Giving a range instead of an *exact* answer provides greater traceability.

Overly optimistic estimates create project failures because dates pass and slip, functionality gets reduced, project budgets get surpassed, and quality suffers (i.e., testing time is cut out). Remember that an estimate is only as good as its least reliable input variable; garbage in equals garbage out. While it is the American way for faster, better, and cheaper solutions, sometimes they are so compelling that management will attempt the impossible through the overly optimistic estimate. The result is that the project will only be done right the second time around [4].

Figure 1: *Sample Project Components*



Estimating During or Before Requirements

When asked to perform an overall project estimate using a requirements-based estimating method, the first step is to decide how many separate (sub)projects are included within the scope of the overall business project if more than one software application is involved. If there is only one software application involved, this step can be skipped. If there is more than one application to be enhanced or developed, each usually has its own set of requirements and will need its own (sub)project estimate[2]. (Usually, each application that undergoes new development or enhancement will be classified and estimated as its own (sub)project, and the overall project effort, cost, and duration can be calculated as combined values. Consider a single overall project with several subprojects: (a) new development project, and (b) two enhancement projects (see Figure 1). Each one would be estimated separately, and the results added together. Additionally, the entire project might also require an estimate for the integration testing of the component subproject pieces. The overall project estimate for cost and effort would be the sum of the subproject estimates, while the duration would depend on task dependencies between and within subprojects[3].

The second step is to identify and estimate the size or impact of the three types of project requirements for each of the subprojects. Consider the fictional subproject 1.

Functional Requirements

The requirements for *what* the software must do might not be defined in enough detail to do functional sizing, but could be approximated [5]. If even one functional component (such as number of entities) is known, an approximation can be done. Several approximation methods are outlined in [6]. Documenting the assumptions about the entities helps to substantiate the estimate[4]. If there is enough data, the functional size approximation can be more accurate and use more accurate techniques. For the two subprojects, each would be assessed based on an approximation of how many function points would be added (new functions as in subproject 3), modified (changed or renovated functions), or removed. The functional size of the subproject is the sum of new plus modified plus removed functions.

Non-Functional Requirements

Assessment of the *ilities* is based on a

comparison to similar projects or a value adjustment factor (which is part of a sizing method such as IFPUG). If the non-functional requirements are unknown, it is best to overestimate their impact as usually they turn out to be more complex than anticipated (e.g., security requirements). Even if estimators and software developers intuitively know that estimates are too low, customers and user managers have an insatiable optimism that maybe, *just this once* it might come true. Time and time again, overly optimistic estimates become self-fulfilling prophecies as dates slip, functionality is reduced, and project budgets are surpassed.

Barry Boehm remarked on the impact of non-functional requirements: "A tiny change in NFRs [non-functional requirements] can cause a huge change in the cost." Boehm cited the tripling of a $10 million project to $30 million when the response time (of a NFR) went from four seconds to one. [8]. It is important to document assumptions for NFRs, especially if project complexity is likely to increase.

Technical Requirements

IT project teams often use a standard suite of development tools and technologies. The technical requirements are usually the least risk prone of the three requirement types – particularly technologies and subject matter are standard. For major changes in technology, further care must be taken to assess this requirements area.

Results should be documented along with the method used, the date, and source documents used for the estimate so that guesstimates and estimates become more traceable and auditable.

Need an estimate for a project that has few or no known input variables? Are there options for an estimator? He or she could attempt these tactics: (a) refuse to do an estimate, (b) delay the estimate repeatedly until requirements are at least partially done, (c) provide a wild guess (which is common), (d) try to find similar completed projects within your own environment and use their actual values, (e) cite *professional* ethics and hide out, or (f) (this is the preferred method) document assumptions and use them together with the estimate (guesstimate) to substantiate the estimation results.

What Can You Do to Improve Project Estimates?

Project estimating can be more auditable and more realistic by applying some of the aforementioned practices. Document as many of your assumptions about the project as you can; revise them and the estimate according to the same/updated assumptions later. Separate, document, and assess (approximation or count) the project into subprojects according to application; address each set of requirements clearly; and objectively split them into the three types: functional, non-functional, and technical. Use an established requirements-based estimating tool or benchmarking database such as COCOMO II or the International Software Benchmarking Standards Group with proven track records for your environment. Label results as preliminary. Teach customers about the estimating process. Educate them that an estimate too early in the life cycle cannot remain fixed throughout the project, nor can it be accurate. And finally, combine the subproject estimates into a single overall project estimate. Present the guesstimate as a range (when information is premature or missing) with a level of accuracy commensurate with what is known about the project at the time (e.g., rounded to the closest $100,000).◆

References

1. The Standish Group. "Latest Standish Group CHAOS Report Shows Project Success Rates Have Improved by 50 Percent." Press Release, 25 Mar. 2003 The Standish Group, <www.standishgroup.com/press/article.php?id=2>.
2. McConnell, Steve. "After the Gold Rush." 2004 Systems and Software Technology Conference, Salt Lake City, UT, 19-22 Apr. 2004.
3. Card, David N. The Role of Measurement in Software Engineering. July 1998.
4. U.S. Army. Insight. Summer, 2003.
5. Hill, Peter R., Ed. Practical Project Estimation. 2nd ed. International Software Benchmarking Standards Group, 2005 <www. isbsg.org>.
6. International Organization for Standardization/International Electrotechnical Commission. ISO/IEC 14143-1:1998 Information Technology – Software Measurement – Functional Size Measurement – Part 1: Definition of Concepts. ISO/IEC <www.jtcl-sc7.org>.
7. ISO/IEC. ISO/IEC 9126 Series of Standards for Measuring Software Quality. ISO/IEC <www.jtcl-sc7.org>.
8. Robertson, Suzanne and James. Preface. Requirements-Led Project Management – Discovering David's Slingshot. By Barry Boehm. Pearson Education, 2005.

Notes

1. *Users* refers to any person, thing, other application, other software, hardware, etc., outside the boundary of the software that has the requirement to send or receive data from the software [6].
2. Even if requirements are collectively listed in a single document, specific requirements will pertain to a specific software application. It is important to divide the requirements among various applications within the overall project to facilitate subproject estimates.
3. The overall duration may not be the summation of the subproject durations; some tasks may proceed concurrently while others may have precedence in other subprojects before they can commence.
4. The one file model or rule of 31 is an approximation technique whereby each identified entity is assumed to have add, change, delete, query, output, and storage function. Using IFPUG FP average values, the total is 31 FP for each entity. For three entities, this equates to 93 FP – or roughly in the range of 100 FP.

About the Author

Carol A. Dekkers is president of Quality Plus Technologies, Inc., a management consulting firm that specializes in helping companies improve their software and systems success. She is a past chair and founder of the American Testing Board, a former president of the International Function Point Users Group, and is active in the Project Management Institute, the American Society for Quality, and the International Organization for Standardization. She is a Certified Management Consultant, a Certified Function Point Specialist, a professional engineer (Canada), an Information Systems Professional, and an International Software Testing Qualifications Testing Board Certified Tester – Foundation Level.

Quality Plus Technologies, Inc.
8430 Egret LN
Seminole, FL 33776
Phone: (727) 393-6048
Fax: (727) 393-8732
E-mail: dekkers@qualityplustech.com